Metaclassifiers: Random forests, rotation forests, bagging, boosting, cascading, hybrid

Universidad Politécnica de Madrid, España

Silvia Arenales Muñoz, Manuel Castillo Sancho, Rubén González Velasco,
Sergio Martos Pariente y Héctor Sanz González.

8 de enero de 2025

Resumen

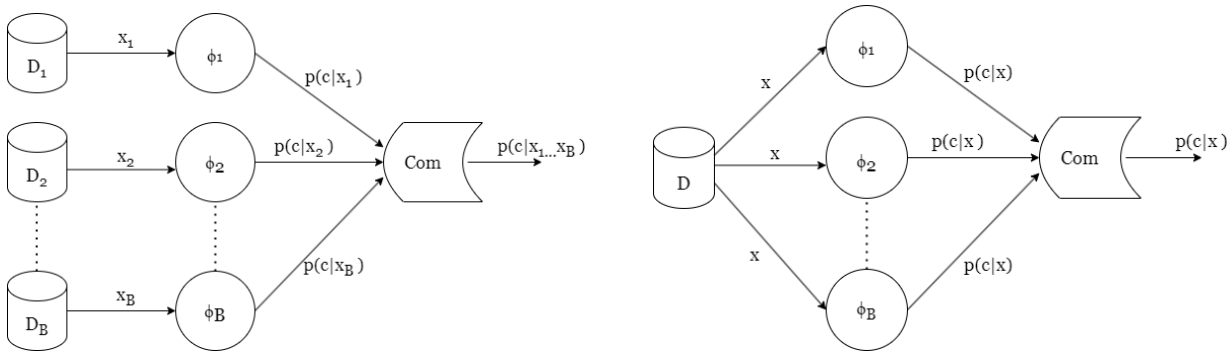
Los metaclasificadores son empleados para mejorar aún más la precisión en la aplicación de algoritmos en un problema determinado. Esta es una de las áreas más activas de investigación en aprendizaje supervisado ya que los ensambles son mucho más precisos que los clasificadores individuales que los componen. Este trabajo presenta un estado del arte sobre los metaclasificadores, centrándose en los siguientes puntos: Random Rorest, Rotation Forest, Bagging, Boosting, Cascading e híbridos.

1. Introducción

Cuando se toman decisiones importantes, se suelen considerar la opinión de varios expertos en lugar de confiar únicamente en un criterio propio. Por ejemplo, se pueden combinar diversos puntos de vista debatiendo o recurriendo a una votación. En el ámbito del aprendizaje automático, un modelo de aprendizaje se considera como un experto cuya opinión puede ser correcta o no, dependiendo de si este es apropiado para el problema en concreto, y de la calidad y cantidad de los datos de entrenamiento de los que se disponga. Sin embargo, según el teorema de *Non free-lunch theorem* de Wolpert et al. (1995), ningún algoritmo de aprendizaje genera el clasificador más preciso en cualquier dominio. Cada algoritmo converge hacia soluciones diferentes y falla en distintas circunstancias. Incluso optimizado para un conjunto de validación, pueden existir muestras en las que no sea preciso, donde otro algoritmo podría funcionar mejor. Por ello, el objetivo es buscar algoritmos que tomen decisiones complementarias.

Se distinguen dos enfoques principales para caracterizar los metaclasificadores, en la Figura 1 se pueden ver los diferentes esquemas. Por un lado, tenemos los metaclasificadores con diferentes espacios de características en donde cada clasificador trabaja con un conjunto distinto de características que proviene de diferentes fuentes o sensores. Cada clasificador produce una probabilidad para una clase específica, basándose en sus características individuales. Estas predicciones son combinadas por un modelo o regla de combinación, que estima la probabilidad conjunta. Por otro lado, están los metaclasificadores con espacio de características común, en donde todos los clasificadores trabajan con el mismo conjunto de características, mismas entradas, enfocándose en aprovechar la diversidad en los métodos, los subconjuntos de datos o los parámetros. Podemos distinguir dentro de este grupo tres variantes: (1) Clasificadores de diferente tipo (híbridos), (2) Clasificadores del mismo tipo pero con diferentes muestras (Bagging, Boosting) y (3) Clasificadores del mismo tipo pero con diferentes parámetros.

Existen diversas formas de configurar metaclasificadores dependiendo de varios criterios: el método para seleccionar los clasificadores base (ya sea priorizando su precisión individual o la diversidad entre ellos), la estructura en la que se organizan (paralela, secuencial o jerárquica) y su naturaleza (si son



(a) Diferentes espacios de características (b) Espacios de características común

Figura 1: Clasificación según espacios de características

clasificadores similares o de tipos diferentes). Además, también influye el enfoque que se adopte, como ajustar el combinador para optimizar su desempeño, garantizar que los clasificadores base logren buenos resultados, equilibrar ambos aspectos, o no priorizar ninguno en particular. En las fuentes de los siguientes autores, se puede encontrar literatura sobre metaclasificadores: Bielza and Larrañaga (2020), Woźniak et al. (2014), Kuncheva (2014) o Rokach (2010).

Sin embargo, la combinación de estos modelos tienen la desventaja de ser difíciles de analizar, ya que pueden estar formados por docenas o cientos de modelos individuales. Aunque su rendimiento es alto, no es sencillo entender de manera intuitiva qué factores están contribuyendo a las mejores decisiones.

Los métodos de conjunto convencionales incluyen métodos basados en bagging, boosting y stacking. Estos métodos han sido bien estudiados en los últimos años y se han aplicado ampliamente en diferentes aplicaciones. Por ejemplo, Bagging se usa ampliamente en clasificación y regresión. Además, Random Forest (una variante de bagging), Random Subspace, AdaBoost (un método de conjunto secuencial) han sido métodos ampliamente estudiados. Es por eso, en este trabajo, nos centraremos en los metaclasificadores más populares. Primero, se abordarán los avances relacionados con Bagging en la Apartado 2.1. A continuación, se explorarán las innovaciones más destacadas en Boosting en la Apartado 2.2. Posteriormente, se analizarán los principales desarrollos en las variantes de Random Forest en la Apartado 2.3, así como en Rotation Forests en la Apartado 2.4. Además, se discutirán los progresos en Cascading en la Apartado 2.5. Por último, se examinarán los avances recientes en metaclasificadores con espacios de búsqueda compartidos pero de naturaleza diversa, en la Apartado 2.7.

2. Estado del Arte

2.1. Bagging

Bagging, acrónimo de *Bootstrap Aggregating*, fue introducido por Breiman (1996) como un método para reducir la varianza de los modelos de aprendizaje supervisado, mejorando la estabilidad y precisión de las predicciones. Este método se basa en el principio de combinar múltiples modelos individuales entrenados con subconjuntos distintos del conjunto de datos original.

Para formalizarlo, Bagging consiste en:

- Generar *muestras bootstrap* del conjunto de datos de entrenamiento. Cada muestra se obtiene mediante muestreo aleatorio con reemplazo. (Boostrapping)
- Entrenar un modelo base independiente con cada una de estas muestras. (Aggregating)
- Combinar las predicciones de los modelos base para obtener la predicción final.(Bagging)

Bagging se formaliza de la siguiente manera:

Definición 1. *Sea $\mathcal{D} = \{x_i, y_i\}_{i=1}^N$ el conjunto de entrenamiento con N observaciones y B el número de modelos base. Bagging construye B modelos $\phi_b(x)$, donde cada modelo ϕ_b es entrenado en una muestra bootstrap $\mathcal{D}_b$ de tamaño N. La predicción final se obtiene mediante diferentes métodos dependiendo del tipo output.*

Algorithm 1: Bagging

Input: *Datos de entrenamiento $\mathcal{D}$, número de modelos base B, modelo base ϕ*
Output: Conjunto de modelos $\{\phi_b\}_{b=1}^B$ y predicción combinada

1 **for** $b = 1, \ldots, B$ **do**
2 | - Generar una muestra bootstrap $\mathcal{D}_b$ a partir de $\mathcal{D}$;
3 | - Entrenar el modelo base ϕ_b con $\mathcal{D}_b$;
4 **Predicción:** Para una nueva instancia x, combinar las predicciones $\{\phi_b(x)\}$ mediante:

- **Clasificación:** Votación mayoritaria: $\hat{y} = \arg\max_y \sum_{b=1}^B \mathbb{I}(\phi_b(x) = y)$.
- **Probabilidades Posteriores:**
  - Opcion 1: Usar las etiquetas predichas (con mayor probabilidad) y votar posteriormente.
  - Opcion 2: Promediar las probabilidades obteniendo nuevas probabilidades para cada clase y elegir aquella con la mayor probabilidad: $\hat{y} = \arg\max_y \frac{1}{B} \sum_{b=1}^B P(y|\phi_b(x))$.
- **Regresión:** Promedio simple: $\hat{y} = \frac{1}{B} \sum_{b=1}^B \phi_b(x)$.

2.1.1. Variantes de Bagging

A lo largo de los años, han surgido variantes de Bagging que buscan optimizar o extender su funcionalidad. La variante más relevante es Random Forest, que se explicará en el Apartado 2.3. A continuación se describen algunas otras:

Pasting. Introducido por Breiman (1999a), Pasting es un enfoque diseñado para manejar bases de datos extremadamente grandes, donde no es posible mantener todo el conjunto de datos en memoria. El procedimiento, denominado *pasting votes together*, tiene los siguientes pasos clave:

Supongamos que, hasta el momento, se han construido k clasificadores. Un nuevo conjunto de entrenamiento de tamaño modesto M se selecciona del conjunto $\mathcal{D}$, ya sea mediante muestreo aleatorio sin reemplazo o muestreo por importancia. El clasificador $(k+1)$ se entrena con este nuevo conjunto y se agrega a los k clasificadores anteriores. La agregación depende del tipo de salida.

Si se utiliza muestreo aleatorio para seleccionar el conjunto de entrenamiento, este se denomina *Rprecinct*, y el procedimiento se llama *pasting Rvotes* (R = aleatorio). Si el muestreo se realiza por importancia, el conjunto de entrenamiento se denomina *Iprecinct*, y el procedimiento se conoce como *pasting Ivotes* (I = importancia). El muestreo por importancia utilizado selecciona más frecuentemente aquellas instancias que tienen mayor probabilidad de ser clasificadas incorrectamente.

Finalmente, una estimación $e(k)$ del error de generalización para la agregación k-ésima se actualiza en cada iteración. El procedimiento de pasting se detiene cuando $e(k)$ deja de disminuir. La estimación de $e(k)$ puede obtenerse utilizando un conjunto de prueba.

Pasting es especialmente útil para el aprendizaje en línea y bases de datos de gran escala, ya que permite manejar datos masivos de manera eficiente sin comprometer la precisión del modelo.

Adaptive Bagging. Propuesto por Breiman (1999b), Adaptive Bagging, o también conocido como Iterative Bagging, es una variante que extiende el bagging tradicional para reducir tanto la varianza como el sesgo del modelo en problemas de regresión a través de un procedimiento iterativo y adaptativo.

El procedimiento utiliza conjuntos de entrenamiento que son muestras bootstrap tomadas del conjunto de entrenamiento original. Esto imita una secuencia de conjuntos de entrenamiento independientes extraídos de la misma distribución subyacente. La implementación del método consta de las siguientes etapas:

1. **Etapas iterativas:** El procedimiento avanza en etapas, cada una consiste en realizar un bagging con un número fijo K de predictores, utilizando los mismos valores de entrada pero alterando los valores de salida. Denotemos los valores de la variable de salida utilizados en la etapa j de bagging como $y_n^{(j)}$, siendo $y_n^{(1)}$ los valores de salida iniciales.

2. **Actualización de los valores de salida:** Durante la etapa j, denotemos por $\hat{y}_{n,k}$ los valores predichos dados por el k-ésimo predictor entrenado. Al final de la etapa j de bagging, los valores de salida se actualizan como:
$$y_n^{(j+1)} = y_n^{(j)} - \text{average}_k \hat{y}_{n,k}, \tag{1}$$
La resta ajusta los valores actuales para compensar las desviaciones entre las predicciones realizadas por el modelo y los valores originales. Si el predictor en la etapa j tiene un sesgo en su predicción, esta corrección ajusta el valor de salida para que las etapas posteriores intenten aprender mejor las relaciones subyacentes. El promedio sobre $\hat{y}_{n,k}$ se toma solo sobre aquellos k tales que la n-ésima instancia no está incluida en el conjunto de entrenamiento k.

3. **Predicción para nuevas entradas:** Si x es un valor de entrada que no está en el conjunto de entrenamiento inicial, entonces el predictor $\phi^{(j+1)}(x)$ después de j etapas se obtiene como:
$$\phi^{(j+1)}(x) = \phi^{(j)}(x) + \text{average}_k \phi_{j,k}(x), \tag{2}$$
donde $\phi_{j,k}(x)$ es el valor predicho para x por el k-ésimo predictor en la etapa j.

Supongamos que hemos repetido el proceso de bagging 100 veces. En cada conjunto bootstrap, aproximadamente el 37 % de las instancias quedan fuera. Por lo tanto, los valores $\hat{y}_{n,k}$ se están promediando sobre aproximadamente 37 conjuntos de entrenamiento. Este promedio se sustrae de $y_n^{(j)}$ para formar los nuevos valores de salida.

Este procedimiento se repite en ciclos sucesivos. La señal para detener las iteraciones es la siguiente: si la suma promedio de cuadrados de los nuevos valores de y es mayor que 1.1 veces el mínimo de la suma promedio de cuadrados de los y en cualquiera de las etapas previas, entonces se detiene el proceso y se utiliza el predictor dado por la etapa final, que tiene la suma promedio de cuadrados más baja.

La razón de esta regla de detención es que la suma promedio de cuadrados de los y es una medida del sesgo residual. Cuando este valor comienza a incrementarse, el proceso de eliminación del sesgo empieza a reflejar ruido. En este caso, se detiene el proceso y se retrocede al valor mínimo.

Subagging (Subsample Aggregating). Introducido por Bühlmann and Yu (2002), el subagging es una variante que utiliza submuestreos en lugar de muestras bootstrap completas para construir predictores agregados. En lugar de tomar muestras con reemplazo del conjunto de datos original, el subagging selecciona subconjuntos sin reemplazo, ofreciendo una reducción significativa en la complejidad computacional. Esto resulta particularmente útil en escenarios con conjuntos de datos muy grandes.

El proceso se define de la siguiente manera:

1. **Selección de submuestras:** Para un conjunto de datos $\mathcal{D} = \{L_1, ..., L_n\}$, se seleccionan submuestras de tamaño m siendo $m < n$, sin reemplazo. Esto genera un conjunto de subconjuntos $\mathcal{S} = \{S_1, ..., S_B\}$, donde cada subconjunto S_i contiene m observaciones.

2. **Construcción de predictores:** Se entrena un predictor base $\phi(S_i)$ en cada submuestra S_i.

3. **Agregación:** Las predicciones de los modelos entrenados en los subconjuntos se combinan para formar el predictor final. En el caso de regresión, esto se realiza mediante un promedio, mientras que para clasificación, se utiliza una votación mayoritaria.

El subagging se formula matemáticamente como anteriormente definimos:

$$\phi_{n;SB}(x) = \frac{1}{B} \sum_{i=1}^{B} \phi(S_i)(x), \tag{3}$$

donde B es el número de submuestras utilizadas.

Bragging (Bootstrap Robust Aggregating). El bragging, acrónimo de *Bootstrap Robust Aggregating*, variante introducida por Bühlmann and Yu (2003) que utiliza estimadores robustos para combinar las predicciones de los modelos base, reduciendo la sensibilidad a valores atípicos. En lugar de calcular la media de las predicciones, como se realiza en el bagging tradicional, el bragging emplea la mediana como agregador:

$$\phi_{n;Brag} = \text{median}(\{\phi_b(x); b = 1, \ldots, B\}). \tag{4}$$

Al usar la mediana, el bragging reduce el impacto de valores extremos en las predicciones, proporcionando un modelo final más robusto frente a ruido y valores atípicos en los datos. Esta propiedad es particularmente útil en escenarios donde algunos clasificadores base pueden generar predicciones anómalas.

Además de la mediana, se han explorado otros estimadores robustos, como el estimador de Huber y el estimador M con recorte de Hampel. Sin embargo, los estudios han demostrado que la mediana ofrece un mejor rendimiento en términos de robustez y estabilidad.

Under-Bagging. El under-bagging, introducido por Wallace et al. (2011), es una variante diseñada específicamente para abordar el problema del desequilibrio de clases en conjuntos de datos en un problema de clasificación binaria. Este problema surge cuando una clase está representada significativamente menos que la otra, lo que puede sesgar los modelos predictivos hacia la clase mayoritarias.

El under-bagging combina los principios de bagging con under-sampling, un muestreo selectivo que submuestrea las clases mayoritarias. Esto tiene como objetivo equilibrar la representación de las clases en cada muestra bootstrap utilizada para entrenar los clasificadores base ϕ_b.

El proceso se desarrolla de la siguiente manera:

1. **Submuestreo de la clase mayoritaria:** Se extrae una muestra bootstrap del conjunto de entrenamiento, asegurando que el número de instancias de la clase mayoritaria sea igualado al de la clase minoritaria. Esto crea un subconjunto balanceado.

2. **Entrenamiento de clasificadores base:** Se entrena cada uno de los clasificadores ϕ_b utilizando cada subconjunto balanceado generado.

3. **Agregación:** Las predicciones de los clasificadores base se combinan utilizando votación mayoritaria.

Online Bagging. El online bagging, introducido por Oza and Russell (2001), es una extensión del bagging diseñada para escenarios en los que los datos llegan de manera continua o en flujos (*data streams*). Este método permite entrenar modelos de manera incremental sin la necesidad de almacenar y procesar múltiples veces el conjunto de datos completo, lo que lo hace especialmente útil para grandes volúmenes de datos.

El bagging tradicional genera muestras bootstrap al seleccionar instancias del conjunto de entrenamiento con reemplazo. En cada muestra, una instancia puede aparecer varias veces, donde el número de ocurrencias siguiendo una distribución binomial. Cuando $N \to \infty$, esta distribución tiende a una distribución de Poisson con media 1. El online bagging aprovecha esta propiedad para simular el muestreo bootstrap en un entorno en línea.

En el online bagging, cada vez que una nueva instancia d llega, se actualizan los modelos base ϕ_b según el siguiente proceso:

1. Para cada modelo base $b = 1, \ldots, B$:

   a) Generar un valor k de acuerdo con una distribución de Poisson con media 1.

   b) Actualizar el modelo base ϕ_b con la instancia d un total de k veces.

2.2. Boosting

Kearns and Valiant (1994) vieron que un clasificador débil puede convertirse en uno fuerte, llevando el error de entrenamiento a cero en un tiempo de ejecución polinomial. A esta técnica se le denominó Boosting.

AdaBoost. Propuesto inicialmente por Freund and Schapire (1997) para clases binarias, AdaBoost tiene la capacidad de ajustarse de manera adaptativa a los errores de los clasificadores débiles. Estos clasificadores débiles son clasificadores cuyo rendimiento es ligeramente mejor que el azar. AdaBoost funciona mejor con clasificadores base débiles, puesto que clasificadores demasiado complejos pueden sobreajustarse. Como referencia, el clasificador débil más utilizado para boosting es CART o C4.5, por su eficiencia, flexibilidad en cuanto a datos (numéricos o categóricos) e interpretabilidad. A diferencia de los algoritmos anteriores, AdaBoost cambia la distribución de muestreo de datos en cada iteración, centrándose más en los ejemplos que fueron clasificados incorrectamente en la ronda anterior. Esta capacidad de adaptación le da su nombre, ADAptive Boosting. AdaBoost destaca como uno de los mejores clasificadores ya que el error de entrenamiento de AdaBoost tiene un límite superior que disminuye de forma exponencial al agregar más clasificadores, siempre que cada clasificador individual tenga un error menor a 0.5.

El clasificador añadido en el paso i se entrena con una muestra del dataset. La distribución inicial del muestreo es uniforme (como en bagging). El paso i actualiza esta distribución, aumentando la probabilidad de las instancias mal clasificadas en el paso $i - 1$, a fin de forzar al algoritmo base (o débil) a centrarse en los ejemplos más difíciles en sucesivas ejecuciones.Al final, los modelos débiles se combinan en una votación ponderada.

Breiman (1998) encaja Adaboost en la familia de algoritmos *ARCING* (Adaptative Resample and CombinING), y muestra que la eficacia de Arcing se debe al *muestreo adaptativo*, no a los detalles técnicos, para mostrarlo, arc-fs y arc-x4 (más simple) se comparan con Bagging (2.1.1).

- **arc-fs** es equivalente a AdaBoost
- **arc-x4**: Cambia algunos detalles técnicos. Actualiza los pesos según las veces que una instancia se clasifica erróneamente. Los clasificadores finales se combinan mediante votación no ponderada.

Algorithm 2: AdaBoost

Input: Conjunto de datos $\mathcal{D} = \{(x^1, c^1), \ldots, (x^N, c^N)\}$, número máximo de clasificadores L
Output: Un metaclasificador ϕ_T

Inicializar los pesos de las instancias: $w_j^1 = \frac{1}{N}$, para $j = 1, \ldots, N$;
for $i = 1, \ldots, L$ **do**
    Dibujar una muestra $\mathcal{D}_i$ de $\mathcal{D}$ usando la distribución $(w_1^i, \ldots, w_N^i)$;
    Entrenar un clasificador débil ϕ_i usando $\mathcal{D}_i$ como conjunto de entrenamiento;
    Calcular el error ponderado: $\varepsilon_i = \sum_{j=1}^{N} w_j^i l_j^i$ ($l_j^i = 1$ si ϕ_i clasifica mal x^j y $l_j^i = 0$ si no);
    if $\varepsilon_i = 0$ *o* $\varepsilon_i \geq 0{,}5$ **then**
        Ignorar ϕ_i, establecer $L = i - 1$ y detenerse;
    else
        Calcular $\alpha_i = \frac{1}{2} \log\left(\frac{1-\epsilon_i}{\epsilon_i}\right)$;
        Actualizar los pesos: $w_j^{i+1} = w_j^i \cdot e^{-\alpha_i(1-l_j^i)}$ para $j = 1, \ldots, N$;
        Normalizar los pesos para que $\sum_{j=1}^{N} w_j^{i+1} = 1$;
La hipótesis final se calcula por votación ponderada: $\phi_T(x) = \text{sign}\left(\sum_{i=1}^{L} \alpha_i \cdot \phi_i(x)\right)$

También en este trabajo, se muestra empíricamente que boosting no suele sobreajustarse, incluso cuando se ejecuta durante miles de iteraciones. Este fenómeno inesperado se debe a que que aún después de que el error de entrenamiento sea nulo, el error de prueba generalmente no aumenta (e incluso puede reducirse) a medida que el modelo se vuelve más complejo. Este fenómeno contradice la Navaja de Occam, que propone que los modelos más simples generalizan mejor. Posteriormente, Bartlett et al. (1998) muestran que este comportamiento se relaciona con la distribución de los márgenes en los ejemplos de entrenamiento.

Es de mencionar que este algoritmo, aparte de la elección del clasificador base, funciona de manera completamente automatizada. Solo hay que ajustar el valor de un parámetro, el número de ejecuciones L, no establece ningún requisito sobre el clasificador base excepto que su error sea menor que el obtenido al azar. Además, una buena propiedad de AdaBoost es su capacidad para identificar valores atípicos.

Aunque AdaBoost es un algoritmo potente, presenta limitaciones importantes. Principalmente, es sensible al ruido y a los outliers, ya que asigna pesos altos a las observaciones mal clasificadas, lo que puede llevar a sobreajuste. Además, tiene dificultades con clases desbalanceadas, problemas de escalabilidad en datasets grandes, y carece de interpretabilidad debido a la combinación de múltiples clasificadores y pesos asignados.

2.2.1. Variantes de Boosting

Las propiedades de AdaBoost han sido estudiadas extensamente, revelando conexiones con diversas áreas teóricas y prácticas. Por ejemplo, las relaciones con la teoría de juegos o la programación lineal reveladas por Schapire (2003), han permitido extender AdaBoost a problemas como la clasificación multiclase, la regularización y la integración de conocimientos previos.

LogitBoost. Collins et al. (2002) mejora la interpretabilidad de AdaBoost. Esto se logra reformulando el problema como una minimización basada en las distancias de Bregman, unificando el boosting y la regresión logística en un marco común. A diferencia de AdaBoost, que ajusta pesos iterativamente, LogitBoost actualiza las probabilidades predichas para cada clase utilizando un modelo aditivo, basado en funciones base que minimizan directamente la pérdida logística.

El trabajo de Bartlett et al. (1998) marcó el inicio de una línea de generalizaciones de AdaBoost centradas en la maximización de diferentes versiones del margen, lo que dio lugar a algoritmos como los propuestos por Demiriz et al. (2002) y Rätsch et al. (2007), que exploran estrategias alternativas para mejorar la generalización del modelo al manejar márgenes de clasificación de forma más flexible.

BrownBoost. Una versión adaptativa del Boost-by-Majority de Freund (1995), introducido en Freund (1999), también se basa en la teoría del margen, pero con un enfoque innovador. Este algoritmo utiliza funciones del margen para asignar pesos dinámicos a los ejemplos, prestando especial atención a aquellos con márgenes pequeños. Mediante ecuaciones diferenciales inspiradas en el movimiento browniano, BrownBoost limita el esfuerzo invertido en ejemplos difíciles o ruidosos, equilibrando así la mejora del margen y la robustez frente al ruido, a diferencia de AdaBoost, que tiende a sobreajustarse en estas situaciones.

Gradient boosting. La idea del gradient boosting se originó en la observación de Breiman (1997) de que el boosting puede interpretarse como un algoritmo de optimización sobre una función de coste adecuada. Posteriormente, Friedman (2001) desarrolló algoritmos explícitos de refuerzo de gradiente de regresión, simultáneamente con la perspectiva más general de functional gradient boosting de Mason et al. (1999). En lugar de cambiar los pesos de los puntos de datos como AdaBoost, el boosting de gradiente se basa en los pseudo-residuos del predictor anterior. Los aprendices débiles h (normalmente árboles de regresión) se entrenan para predecir los pseudo-residuos de los clasificadores ϕ_i, que por diseño, son el gradiente negativo de la función de pérdida respecto a las predicciones. El modelo final es:

$$\phi_T(x) = \phi_0(x) + \sum_{t=1}^{T} \nu \cdot h_t(x)$$

Para regularizar el modelo y reducir el sobreajuste, el modelo débil se multiplica por un valor pequeño llamado reducción (por ejemplo, $\nu = 0{,}1$) antes de agregarse al modelo fuerte.

Stochastic Gradient Boosting (SGB or Gradient Boosting Machines). Continuando su trabajo, Friedman (2002) desarrolló SGB. La diferencia con lo anterior es que en SGB cada árbol se entrena utilizando un subconjunto aleatorio de datos, lo que introduce un elemento de aleatoriedad que ayuda a reducir la varianza y mejorar la generalización.

Existen implementaciones muy optimizadas de algoritmos de Gradient Boosting y SGB, utilizando comúnmente árboles de regresión como modelos base (los llamados Gradient Boosted Decission Trees (GBDT)), que incorporan muchas modificaciones y heurísticas inteligentes. Algunas de ellas son:

- **XGBoost** (Extreme gradient boosting) de Chen and Guestrin (2016), conocido por su velocidad y rendimiento, presentan optimizaciones como el uso de bloques paralelos.
- **LightGBM** (Light Gradient Boosting Machine) de Ke et al. (2017) introduce técnicas como Gradient-Based One-Side Sampling (GOSS) y Exclusive Feature Bundling (EFB) para mejorar la eficiencia y reducir el consumo de memoria.
- **CatBoost** (Categorical Boosting) dessarrollado por Prokhorenkova et al. (2018), maneja de manera efectiva las variables categóricas y mitiga el sobreajuste mediante el uso de técnicas como el Ordered Boosting.

En Bentéjac et al. (2021) y en Florek and Zagdański (2023) se realizan análisis comparativos de estas implementaciones. No existe una solución universal; la elección óptima depende de las características específicas del conjunto de datos y de los requisitos del proyecto.

Boosting Multiclase En problemas binarios, AdaBoost optimiza una función de pérdida basada en errores de clasificación. Sin embargo, en problemas multiclase, la generalización de esta función no es trivial. Los métodos como AdaBoost.M1 y AdaBoost.M2, propuestos por Freund and Schapire (1997), utilizan enfoques basados en métodos de corte en clases múltiples, donde se entrenan clasificadores binarios para cada clase en comparación con el resto. Posteriormente, Hastie et al. (2009) propusieron SAMME (*Stagewise Additive Modeling using a Multiclass Exponential loss function*) para simplificar y mejorar la extensión multiclase de AdaBoost, eliminando la necesidad de que los clasificadores débiles superen el azar en todas las clases. Mukherjee and Schapire (2010) destacan que algoritmos como AdaBoost.MM, derivados utilizando métodos teóricos de juegos, son eficaces para el boosting multiclase, en particular porque utilizan una condición mínima de aprendizaje débil (minimal weak learning condition).

En el caso de Gradient Boosting multiclase, la extensión también presenta desafíos relacionados con el cálculo de las funciones de pérdida adecuadas. Friedman (2001) en su trabajo original describe Gradient Boosting con funciones de pérdida que se pueden extender a problemas multiclase, como el logloss multinomial. Implementaciones populares como XGBoost, LightGBM y CatBoost soportan directamente problemas multiclase con métodos como este.

Entre los algoritmos más utilizados destacan XGBoost, LightGBM y CatBoost, debido a su robustez, escalabilidad y soporte nativo para clasificación multiclase. Además, enfoques teóricos como LogitBoost y variantes como SAMME son relevantes en contextos específicos. Otras variantes que componen el estado del arte son: TFBT (*TensorFlow Boosted Trees*) de Ponomareva et al. (2017), GBDT-MO (*Gradient Boosted Decision Trees for Multiple Outputs*) de Zhang and Jung (2020), y C-GB (*Condensed Gradient Boosting*) de Emami and Martínez-Muñoz (2024).

Otras Variantes. La investigación actual en Boosting se enfoca en reducir costos computacionales, mejorar su estructura, desarrollar marcos teóricos unificados y explorar técnicas avanzadas de regularización. Entre los avances recientes destacan: el uso de árboles oblicuos dispersos optimizados mediante TAO de Gabidolla and Carreira-Perpiñán (2022), el algoritmo TRBoost (*Trust-region Boosting*) basado en modelos cuadráticos restringidos de Luo et al. (2023), el método EFLSGB (*Ensemble of Fast Learning Stochastic Gradient Boosting*) que acelera significativamente los tiempos de cómputo de Li et al. (2022), el marco teórico *Distributed Gradient Boosting Forest* (DGBF) que unifica Random Forest y Gradient Boosting de Delgado-Panadero et al. (2023), y nuevos criterios de parada basados en la magnitud de los coeficientes de regularización de Fdez-Díaz et al. (2023).

2.3. Random Forest

Ho (1995) propuso por primera vez el concepto de bosque aleatorio de decisión. Específicamente, propuso construir un clasificador basado en árboles de decisión. La cantidad de árboles de decisión contenidos en este clasificador puede expandirse indefinidamente, y estos árboles de decisión se combinan de manera complementaria o ponderada para construir un nuevo clasificador, conocido como el bosque aleatorio de decisión mencionado. Posteriormente, Breiman (1996) propuso el algoritmo bagging, explicado anteriormente en el Apartado 2.1. Ho (1998) propuso además el algoritmo de subespacio aleatorio. Este algoritmo extrae muestras del espacio de características original mediante muestreo bootstrap y construye múltiples subespacios de características. Por tanto, Breiman (2001), integró el algoritmo bagging, el algoritmo de subespacio aleatorio y los árboles de clasificación y regresión (CART), proponiendo así el Random Forest (RF), con el fin de construir una colección de árboles de decisión con varianza controlada.

Para una mayor comprensión, es interesante recurrir a la definición definida por Breiman (2001):

Definición 2. *Un Random Forest es un clasificador que consiste en una colección de clasificadores con estructura de árbol* $h(x, \theta_k), k = 1, ...,$ *donde los* θ_k *son vectores aleatorios independientes e idénticamente distribuidos, y cada árbol emite un voto unitario por la clase más popular entrada* x*.*

Es decir, aquí se utilizan los RF como clasificadores base: cada árbol se construye a partir de un vector aleatorio. Cada árbol es construido usando el siguiente algoritmo 3.

Algorithm 3: Random Forest para Clasificación

Input: N(número de casos prueba), M(número de variables), B(número de árboles)
Output: Conjunto de árboles de decisión $\{T_b\}_1^b$

1 $m = \sqrt{M}$
2 **for** b=1, . . . , B **do**
3 (**i**) Extraer una muestra de bootstrap de tamaño N (con reemplazo) del conjunto de entrenamiento. Usar las instancias no seleccionadas (*out-of-bag*) para evaluar el error.
4 (**ii**) Crecer un árbol de decisión binario T_b para los datos de bootstrap de la siguiente manera, hasta que se cumpla el criterio de detención:
5 - Seleccionar aleatoriamente m variables del conjunto de características.
6 - Evaluar todas las divisiones posibles sobre estas m variables.
7 - Dividir el nodo usando la mejor variable y su valor óptimo de división (métrica Gini o ganancia de información).
8 **end**

(**i**) Se realiza *bootstrap sampling*, aleatoriedad en las instancias. Al igual que el bagging cada árbol se entrena con un subconjunto diferente del conjunto de datos original, obtenido mediante muestreo con reemplazo. De esta manera se reduce el riesgo de sobreajuste al crear diversidad entre los árboles base. Sin embargo, algunos datos pueden no ser seleccionados en ninguna muestra, lo que permite calcular métricas como el error fuera de bolsa (out-of-bag error).

(**ii**) Mediante *Random subset method*, se seleccionan aleatoriamente características en cada nodo del árbol. Es decir, en cada nodo, en lugar de considerar todas las características para encontrar la mejor división, se selecciona un conjunto aleatorio de características (m) de tamaño fijo. La división del nodo se elige únicamente entre estas m características seleccionadas. Donde $m = \sqrt{M}$, siendo M el número total de características. El hiperparámetro B se elige en función del objetivo del problema. Esta diferencia que tiene este algoritmo frente a Bagging, hace que haya diversidad entre los clasificadores (árboles), reduciendo así la correlación. Es especialmente útil cuando hay más características redundantes o irrelevantes, aunque, en caso de pocas características útiles, puede que el árbol no las seleccione. Los criterios para seleccionar la división en cada nodo y los puntos de división se basan en métricas como el índice de Gini o la entropía de información, típicos de los árboles CART (Classification and Regression Trees) utilizados como clasificadores base. Los criterios de parada de cada árbol dependen por diferentes hiperparámetros como profundidad máxima del árbol, tamaño mínimo de hoja, número mínimo de instancias para dividir un nodo, pureza del nodo o complejidad computacional típicamente. Esto asegura que los árboles no crezcan de forma excesiva, manteniendo un equilibrio entre complejidad y generalización.

Una vez construidos los árboles, sus predicciones se combinan mediante votación mayoritaria (*hard voting*) en problemas de clasificación. Este enfoque aprovecha la diversidad entre los árboles base para mejorar la precisión y la robustez del modelo final.

Sin embargo, desde la propuesta de Breiman (2001) se han desarrollado numerosas variantes, cada una adaptada para resolver problemas específicos o mejorar ciertos aspectos del algoritmo original. A continuación, se describen algunas de las variantes más comunes y sus características.

2.3.1. Variantes de Random Forest

En el RF tradicional, se suele utilizar el método de votación por mayoría para combinar múltiples arboles de decisión, el cual ha sido mejorado por numerosos investigadores. Por ejemplo:

- **Diferentes mecanismos de votación.** Tripoliti et al. (2013) propusieron modificaciones al mecanismo de votación basadas en la selección de características, agrupamiento (clustering), vecinos más cercanos y técnicas de optimización.
- **Potential Nearest Neighbor.** Li et al. (2018) desarrollaron un nuevo mecanismo de votación basado en el vecino más cercano potencial (Potential Nearest Neighbor), reemplazando el mecanismo de votación por mayoría tradicional para evitar la pérdida de información causada por las muestras fuera de bolsa (out-of-bag).

Como se mencionó anteriormente, el clasificador base del RF es CART, el cual tiene un rendimiento general al resolver problemas individualizados. Muchos investigadores han estudiado este enfoque desde múltiples perspectivas:

Desde la perspectiva de los criterios de división de nodos en los árboles de decisión:

- **Extremely Randomized Trees (Extra-Trees).** Introducido por Geurts et al. (2006), Extra-Trees es una variante de RF que añade más aleatorización. En lugar de buscar la mejor división de características, las divisiones en cada nodo se seleccionan aleatoriamente. Además utiliza todo el conjunto de entrenamiento en lugar de muestras bootstrap, lo que reduce el sesgo. Esto hace que reduzca aún más el riesgo de sobreajuste y acelere el entrenamiento. Aunque esto a su vez, hace que pierda interpretabilidad.
- **Oblique Random Forest.** Es introducido por Menze et al. (2009) en donde se usa hiperplanos oblicuos en lugar de cortes ortogonales en la división de nodos, permitiendo divisiones más complejas y ajustadas, y así combinar múltiples características. Teniendo así,

Desde la perspectiva del proceso de construcción de los árboles de decisión:

- **Weighted Random Forest.** Esta variante, introducida por Chen et al. (2004), asigna pesos diferentes a cada árbol en función de su rendimiento individual, mejorando el modelo final en datos con ruido o desbalanceo de clases. En el método tradicional, todos los árboles contribuyen por igual al modelo, pero en wRF, los árboles más precisos reciben mayor peso durante la agregación de predicciones. Esta técnica se diseñó para abordar problemas en datos de alta dimensionalidad, como los estudios genéticos, donde las interacciones entre variables pueden ser difíciles de detectar.

Hasta este punto, hemos analizado variantes del RF que introducen modificaciones en pasos específicos del algoritmo. Sin embargo, algunas variantes más recientes destacan por alterar múltiples pasos del proceso, redefiniendo tanto la construcción de los árboles como la forma en que se utilizan las muestras de datos. Entre estas variantes se encuentran:

Denil14. Denil et al. (2014) propusieron una nueva variante de RF, denominada Denil14. Este divide el conjunto de entrenamiento en dos partes: una parte estructural y una parte de estimación. La parte estructural se utiliza únicamente para entrenar los puntos de división, mientras que la parte de estimación se usa para determinar las etiquetas de los nodos hoja. Además, en cada nodo, el tamaño del subespacio de características se selecciona basándose en una distribución de Poisson, y se buscan la característica y el valor de división óptimos a partir de muestras de la parte estructural que han sido preseleccionadas. Esta variante ha demostrado tener consistencia débil y puede ser utilizada para problemas de clasificación.

Bernoulli RF (BRF). Inspirados en el modelo Denil14, Wang et al. (2017) propusieron un nuevo modelo llamado Bernoulli RF (BRF), basado en la distribución de Bernoulli. Similar a Denil14, BRF divide el conjunto de datos en una parte estructural y una parte de estimación, entrena los puntos de división utilizando la parte estructural y determina las etiquetas de los nodos hoja con la parte de estimación. Sin embargo, BRF introduce dos distribuciones de Bernoulli en cada nodo: una para determinar si el tamaño del subespacio de características será 1 o $\sqrt{M}$, y otra para decidir si el valor de división de cada característica candidata se selecciona aleatoriamente o utilizando el criterio óptimo de división. Finalmente, demostraron que BRF también tiene consistencia débil, pero está más cerca de BreimanRF y tiene mejor rendimiento que las variantes previas de RF con consistencia débil. BRF puede utilizarse tanto para clasificación como para regresión.

Double random forest. Esta variante, Double Random Forest (DRF), introducida por Han et al. (2020), utiliza el conjunto completo de datos de entrenamiento para construir árboles de decisión, a diferencia del RF, que utiliza muestras bootstrap con menos instancias únicas (al rededor de 63,2 % del total). Esto permite al DRF generar árboles más grandes, reduciendo el sesgo en las predicciones. El muestreo bootstrap en DRF se aplica solo momentáneamente en cada nodo intermedio (incluido el nodo raíz) para determinar las reglas de partición, seleccionando también subconjuntos aleatorios de características, como en RF. Una vez definida la partición, se restauran los datos originales del nodo y las instancias se distribuyen hacia los nodos hijos. Este enfoque combina más datos y mayor aleatoriedad, logrando árboles más robustos y un ensamblado más diverso.

Por razones de espacio y debido a la amplia diversidad de variantes del Random Forest, resulta imposible abordar exhaustivamente todas las propuestas derivadas de este algoritmo. Sin embargo, se destacan algunos trabajos que representan avances interesantes y específicos en diferentes contextos. Entre ellos se encuentran: el Online Random Forest Saffari et al. (2009), que permite la actualización dinámica del modelo; el Oblique Random Forest de Menze et al. (2009), que utiliza hiperplanos oblicuos para mejorar las divisiones; el Aggregating Random Forest Trees by Elastic Net (RARTEN) propuesto Farhadi et al. (2024), que combina elasticidad y robustez. Estas variantes reflejan la versatilidad del Random Forest y su capacidad para adaptarse a desafíos específicos, consolidándose como un algoritmo fundamental en la ciencia de datos.

2.4. Rotation Forests

Rotation Forests(RoF) método introducido por Rodriguez et al. (2006), diseñado para generar conjuntos de clasificadores con alta precisión y diversidad. Se basa en la idea de generar diversidad entre los árboles del ensamble mediante la aplicación de transformaciones lineales aleatorias a los datos de entrada. Esto se logra a través de técnicas como Análisis de Componentes Principales (PCA) de Wold et al. (1987) para rotar los datos en un espacio de características modificado. El objetivo es maximizar la precisión individual y la diversidad entre clasificadores.

A diferencia de Random Forest, donde se seleccionan subconjuntos de características de forma aleatoria para cada árbol, en Rotation Forest se generan nuevas características combinando las originales usando transformaciones lineales. Rotation Forest supera a Bagging en precisión y diversidad, ya que utiliza todo el conjunto de datos y transforma las características; y a su vez, utiliza transformaciones de características, lo que introduce combinaciones lineas más ricas y aumenta la diversidad en comparación con Random Forest. El proceso se muestra en el algoritmo 4.

2.4.1. Variantes de Rotation Forest

RotBoost. Después del trabajo pionero de Rotation Forest, Zhang and Zhang (2008) y Zhang and Zhang (2010) propusieron dos variaciones del algoritmo. Presentaron RotBoost como una nueva técnica de ensamblado que combina Rotation Forest y AdaBoost. AdaBoost se integra en el método RotBoost

Algorithm 4: Rotation Forest

Input: N(número de casos prueba), M(número de variables), B(número de árboles)
Output: Conjunto de árboles de decisión $\{T_b\}_1^b$

$m = \sqrt{M}$
for b=1, . . . , B **do**
  Dividir las M variables en m subconjuntos $F_{i,j}$ para $j = 1, ..., m$.
  Para cada subconjunto ($F_{i,j}$):
    Seleccionar una muestra bootstrap del conjunto de datos asociada al subconjunto $F_{i,j}$.
    Aplicar PCA a la muestra seleccionada para generar una matriz de transformación $C_{i,j}$.
  Construir la matriz de rotación R_b uniendo las matrices de transformación $C_{i,j}$ de los subconjuntos.
  Transformar el conjunto de datos X usando R_b para obtener $X'_b = X \cdot R_b$.
  Construir un árbol T_b con X'_b y Y aplicando:
    Selección aleatoria de m variables en cada nodo.
    Evaluar todas las divisiones posibles sobre las m variables seleccionadas.
    Dividir el nodo usando la mejor variable y su valor óptimo de división (métrica Gini o ganancia de información).
end

para aprovechar las ventajas de ambos enfoques, logrando una reducción significativa del error de predicción en comparación con usar AdaBoost o Rotation Forest de forma individual. Los resultados experimentales mostraron que RotBoost supera tanto a Rotation Forest como a AdaBoost en muchos casos.

Ensemble Feature Forest (FFF) de Zhang and Suganthan (2014) utilizaron PCA y LDA (Análisis Discriminante Lineal) de Ye et al. (2004), para entrenar un bosque en ensamblado que mapea los datos originales en diferentes espacios rotacionales y luego concatena las características proyectadas junto con las características originales en un espacio de mayor dimensionalidad en el nodo raíz.

2.5. Cascading

Los clasificadores en cascada representan una estrategia eficaz dentro del ámbito de los clasificadores multietapa. En esencia, pueden considerarse como una extensión de estos últimos, ya que comparten la idea de procesar las instancias de manera progresiva, pasando por múltiples etapas de clasificación. En el articulo de Pudil et al. (1992), se introdujo la posibilidad de incorporar una opción de rechazo en cada etapa de clasificación para clasificadores multietapa. Esto implica que, en lugar de tomar una decisión final en cada etapa, se permite que las instancias con resultados ambiguos sean transferidas a las etapas posteriores, donde clasificadores adicionales pueden realizar un análisis más detallado. Lo cual dio lugar a la idea en las que se basan los clasificadores en cascada.

Sin embargo, no sería hasta la publicación de Alpaydın and Kaynak (1997) que se formalizaría el concepto de los clasificadores en cascada tal como los conocemos hoy. La motivación principal detrás de este enfoque es reducir el costo computacional asociado al uso de clasificadores complejos y mejorar el rendimiento general del sistema de clasificación. En lugar de emplear un clasificador único y sofisticado para todas las instancias, los clasificadores en cascada adoptan una estrategia secuencial, procesando las instancias a través de una serie de etapas.

El proceso comienza con clasificadores iniciales que son simples y computacionalmente rápidos. Estos clasificadores están diseñados para resolver con alta confianza las instancias que son fáciles de clasificar. Aquellas instancias que no pueden ser resueltas de manera concluyente en estas primeras etapas son transferidas a clasificadores más complejos en etapas posteriores. Este enfoque jerárquico

garantiza que solo una fracción de las instancias (las que presentan mayor dificultad) requiera análisis detallados por parte de los clasificadores más costosos.

Definición 3. *Sea $\mathcal{D} = \{x_i, y_i\}_{i=1}^{N}$ el conjunto de entrenamiento con N observaciones y L el número de niveles en la cascada. Los metaclasificadores en cascada construyen una serie de clasificadores $\phi\ell(x)$, donde cada clasificador ϕ_ℓ del nivel ℓ es entrenado en un subconjunto $\mathcal{D}_\ell \subseteq \mathcal{D}$ que contiene las instancias no clasificadas con confianza por los niveles anteriores. La predicción final se obtiene evaluando de manera secuencial cada nivel ℓ, con modelos más complejos en niveles superiores manejando casos más difíciles.*

A continuación se presenta el pseudocódigo:

Algorithm 5: Cascading

Input: *Datos de entrenamiento $\mathcal{D}$, número de niveles de la cascada L, modelos base $\{\phi_\ell\}_{\ell=1}^{L}$, umbral de confianza τ_ℓ para cada nivel ℓ*

Output: Conjunto de modelos $\{\phi_\ell\}_{\ell=1}^{L}$ y predicción final

for $\ell = 1, \ldots, L$ **do**
- **if** $\ell = 1$ **then**
  - Entrenar ϕ_1 con el conjunto completo $\mathcal{D}_1 = \mathcal{D}$;
- **else**
  - Seleccionar las instancias de $\mathcal{D}$ no clasificadas con confianza por los niveles anteriores:

$$\mathcal{D}_\ell = \{(x_i, y_i) \in \mathcal{D} : P(y_i \mid \phi_{\ell-1}(x_i)) < \tau_{\ell-1} \text{ o } \phi_{\ell-1}(x_i) \neq y_i\}.$$

  - Entrenar ϕ_ℓ con $\mathcal{D}_\ell$;

Predicción: Para una nueva instancia x: **for** $\ell = 1, \ldots, L$ **do**
- Calcular la predicción $\phi_\ell(x)$ con su confianza $P(y \mid \phi_\ell(x))$;
- **if** $P(y \mid \phi_\ell(x)) \geq \tau_\ell$ **then**
  - Retornar $\phi_\ell(x)$ como la predicción final;

Si ningún nivel clasifica con confianza: Usar un clasificador no paramétrico (por ejemplo, k-NN) para manejar las instancias restantes.

2.6. Variantes Cascading

Los clasificadores en cascada han experimentado una evolución significativa a lo largo del tiempo, dividida principalmente en dos enfoques diferenciados.

Por un lado, estos clasificadores han demostrado ser altamente efectivos en tareas de clasificación de imágenes, lo que ha fomentado su desarrollo en conjunto con mejoras relacionadas con el uso de características específicas para el reconocimiento visual. Sin embargo, es importante destacar que estos avances no están intrínsecamente vinculados a los clasificadores en cascada en sí mismos, sino a las características empleadas en dichos procesos. En este contexto, se abordará la variante inicial que sentó las bases para múltiples modificaciones y optimizaciones posteriores de los clasificadores en cascada.

Por otro lado, aunque se ha realizado un volumen considerablemente menor de trabajo en comparación con los esfuerzos centrados en las características, también se han llevado a cabo investigaciones dirigidas a mejorar la arquitectura de los clasificadores en cascada para incrementar su eficiencia. En este documento, se exponen los avances metodológicos más significativos en las técnicas para trabajar con estos clasificadores.

Cascada de Harr. El método de Viola and Jones (2001) introduce un clasificador en cascada optimizado específicamente para la detección rápida de objetos. Este enfoque se distingue por incorporar innovaciones clave como las imágenes integrales, que permiten cálculos más eficientes, y la selección de características a través de *AdaBoost*, un algoritmo que identifica iterativamente las características más discriminatorias para el proceso de clasificación.

En este método, los clasificadores débiles se construyen en torno a una única característica f_j, y cada uno de ellos se define mediante una función $h_j(x)$, formalizada como:

$$h_j(x) = \begin{cases} 1, & \text{si } p_j f_j(x) < p_j \theta_j, \\ 0, & \text{en otro caso,} \end{cases}$$

donde θ_j representa un umbral y p_j la polaridad. *AdaBoost* combina estos clasificadores débiles seleccionados en un único *clasificador fuerte*, que luego se organiza en una jerarquía en cascada.

Esta estructura permite una evaluación eficiente de grandes volúmenes de datos, descartando rápidamente las regiones irrelevantes y focalizándose en las más prometedoras. En conjunto, estos avances tecnológicos garantizan un rendimiento escalable y optimizado para la detección rápida y efectiva de objetos.

Boosting Chain. En el esquema tradicional propuesto por Viola y Jones, cada nivel de la cascada es entrenado de forma independiente, lo cual limita la capacidad de aprovechar la información acumulada en las etapas previas. Para superar esta limitación, Xiao and Zhu (2003) introducen el concepto de *boosting chain*, que extiende las cascadas tradicionales integrando explícitamente el conocimiento histórico entre niveles consecutivos. En este modelo, cada clasificador de la cadena utiliza las predicciones acumuladas de los clasificadores anteriores como base para su entrenamiento, mejorando así la coherencia y el rendimiento global.

Matemáticamente, el clasificador $\Phi_i(x)$ en el esquema *boosting chain* está definido como:

$$\phi_i(x) = \text{sign}\left(\sum_{k=1}^{i-1}\sum_{t=0}^{m_k} \alpha_{t,k} h_{t,k}(x) - b_k\right),$$

donde:

- $\phi_i(x)$ representa el clasificador i-ésimo, que toma decisiones basadas en la información histórica de los niveles previos (1 a $i-1$) y la capa actual.
- sign$(\cdot)$ es una función que devuelve 1 si el argumento es positivo (detección positiva) y 0 en caso contrario. Su finalidad es simplificar el calculo para obtener una salida binaria.
- $h_{t,k}(x)$ es el t-ésimo clasificador débil en la capa k, diseñado para evaluar una característica específica.
- $\alpha_{t,k}$ es el peso asignado al t-ésimo clasificador débil en la capa k, ajustado según su importancia.
- m_k indica el número total de clasificadores débiles en la capa k, que define la complejidad de dicha capa.
- b_k es el umbral que ajusta la decisión final de cada capa; regiones que no alcanzan este valor son descartadas como irrelevantes.

Además, el esquema ajusta dinámicamente los pesos de las instancias en función de su dificultad para ser clasificadas correctamente. Las muestras mal clasificadas o con baja confianza en niveles anteriores reciben mayores pesos en las etapas posteriores, dirigiendo el entrenamiento hacia las áreas más complejas del espacio de características.

Por último, el *boosting chain* emplea un modelo de optimización lineal para equilibrar la tasa de detección con la tasa de falsos positivos, maximizando la eficacia del sistema mientras se mantienen bajos los costos computacionales y los errores en la clasificación.

Soft Cascade. El *Soft Cascade* es una variante estrechamente relacionada con el *boosting chain* y fue propuesta por Bourdev and Brandt (2005), para mejorar la flexibilidad y el rendimiento de los sistemas de clasificación. En este enfoque, cada etapa genera un valor escalar acumulativo $H_k(x)$, definido como:

$$H_k(x) = \sum_{t=1}^{m_k} c_t(x)$$

donde $c_t(x)$ es la salida ponderada de los clasificadores débiles correspondientes a la etapa k.

Una característica clave del *Soft Cascade* es que las decisiones de rechazo se toman en función de un umbral acumulativo r_k después de cada etapa:

$$\text{Si } H_k(x) < r_k, \text{ rechazar muestra como no-positiva.}$$

Este mecanismo suaviza el proceso de decisión, permitiendo que una muestra 'falle' en una etapa sin ser descartada de inmediato, lo que mejora la tolerancia a errores intermedios.

Otra característica distintiva del *Soft Cascade* es la separación entre el entrenamiento del clasificador y su etapa de calibración. Una vez que el modelo ha sido entrenado, los umbrales r_k y el orden de las etapas se optimizan de manera independiente con el objetivo de equilibrar tres aspectos fundamentales: la tasa de detección, la tasa de falsos positivos y el tiempo de ejecución.

Este proceso de optimización se lleva a cabo explorando la superficie ROC (*Receiver Operating Characteristic*), la cual encapsula las interdependencias entre dichos factores. Mediante la proyección de puntos sobre esta superficie, es posible evaluar múltiples configuraciones y seleccionar aquellas que minimicen los falsos positivos o maximicen la eficiencia del sistema. Esto se logra ajustando dinámicamente los umbrales de las etapas, permitiendo un balance preciso entre la precisión y la velocidad del clasificador.

A traves de este enfoque se permite realizar ajustes rápidos en el rendimiento del modelo sin la necesidad de reentrenarlo completamente.

Gracias a estas características, el *Soft Cascade* ofrece una mayor flexibilidad y una toma de decisiones más robusta gracias al uso de trazas acumulativas, reduciendo la sensibilidad a fallos en etapas individuales.

FCBoost. Este método, que fue introducido por Saberian and Vasconcelos (2010), presenta avances significativos en la optimización automática de su configuración. En particular, los siguientes elementos se determinan de manera dinámica durante el proceso:

- **Número de etapas:** Este se ajusta automáticamente con base en la contribución de cada etapa al riesgo total del modelo, logrando así una estructura adaptativa y eficiente.
- **Número de clasificadores débiles por etapa:** Se calcula de forma iterativa, añadiendo clasificadores débiles hasta alcanzar un balance óptimo entre precisión y complejidad computacional.
- **Umbrales de rechazo:** Se optimizan conjuntamente con el entrenamiento, buscando maximizar la eficiencia computacional sin comprometer la precisión del sistema.
- **Muestreo de ejemplos negativos:** Se implementa a través de la técnica de *bootstrapping*, que actualiza dinámicamente el conjunto de entrenamiento al reemplazar una parte de los ejemplos negativos con aquellos identificados como falsos positivos en la cascada.

La cascada emplea coeficientes acumulativos ($T_{1,k}$ y $T_{2,k}$) que determinan la contribución de cada etapa al resultado final. Esta modularidad garantiza que las muestras se procesen únicamente hasta el nivel necesario, maximizando la eficiencia al descartar tempranamente aquellas muestras irrelevantes.

Un componente clave es el concepto de **predictores neutrales**, que simplifica el diseño de cascadas optimizadas. Debido a que replicar la última etapa ($f_m(x)$) no modifica la regla de decisión de la cascada:

$$H[F(f_1, \ldots, f_m, f_m)] = H[F(f_1, \ldots, f_m)],$$

se considera que este último predictor no agrega información significativa. Este enfoque refuerza el objetivo de maximizar el rendimiento global del sistema evitando redundancias en las etapas finales.

2.7. Clasificadores Híbridos

El enfoque híbrido , también conocido como clasificadores de conjuntos heterogéneos, es una técnica que combina múltiples clasificadores individuales de diferentes tipos para mejorar el rendimiento general de la clasificación. Aprovechando las virtudes de los diferentes algoritmos base a su diversidad capturando diferentes aspectos de los datos, compensando sus debilidades, ya que los errores de un clasificador base pueden ser compensados por las predicciones correctas de otro clasificador.

NBTree. El algoritmo NBTree, desarrollado por Kohavi et al. (1996), es un enfoque híbrido el cual combina árboles de decisión y clasificadores Naive Bayes para mejorar la precisión de la clasificación, especialmente en bases de datos grandes. Este enfoque busca aprovechar la segmentación de de datos de los arboles de decisión lo que facilita la tarea de clasificación y la acumulación de evidencias de múltiples atributos de Naive Bayes lo que puede aumentar la precisión.

Para formalizarlo, NBTree consiste en:

- Evaluación de la utilidad de la división: Para cada atributo, NBTree evalúa la utilidad de realizar una división en ese atributo. La utilidad se mide utilizando la precisión de validación cruzada de un clasificador Naive Bayes en el nodo actual y en los nodos hijos resultantes de la división. Si la precisión del clasificador Naive Bayes en el nodo actual es similar o mejor que la precisión obtenida al dividir los datos, entonces no se realiza la división y se crea un clasificador Naive Bayes en el nodo actual.

- Selección del mejor atributo: Si la división es útil, se selecciona el atributo que maximiza la utilidad de la división. Los criterios de selección de atributos comunes en los árboles de decisión, como la ganancia de información o la reducción de la varianza, también se pueden utilizar en NBTree.

- División recursiva: El conjunto de datos se divide en subconjuntos según el atributo seleccionado. El algoritmo NBTree se aplica recursivamente a cada subconjunto, creando nodos hijos en el árbol de decisión

- Creación de clasificadores Naive Bayes: En las hojas del árbol de decisión, donde la utilidad de la división ya no es significativa, se crean clasificadores Naive Bayes. Estos clasificadores se utilizan para clasificar las instancias que llegan a esas hojas.

Lazy Bayesian Rules El algoritmo Lazy Bayesian Rules Zheng and Webb (2000), es un enfoque híbrido el cual combina el clasificador bayesiano Naive Bayes que asume la independencia condicional de los atributos y Lazy learning construye un modelo a medida que tiene que clasificar una nueva instancia, en el momento de la clasificación solo utiliza las instancias que son mas relevantes.
Para formalizarlo, LBR consiste en:
El algoritmo LBR construye una regla bayesiana específica para cada instancia de prueba. Esta regla

se compone de un antecedente y un consecuente. El antecedente es una conjunción de pares atributo-valor, y el consecuente es un clasificador Naive Bayes local.
La construcción de la regla se basa en la idea de que algunos atributos pueden ser más importantes que otros para clasificar una instancia particular. LBR intenta identificar un subconjunto de atributos que minimicen el error de estimación del clasificador Naive Bayes. Este proceso se realiza de forma iterativa, eliminando en cada paso el atributo que se cree que tiene el mayor impacto negativo en la precisión del clasificador.
El clasificador Naive Bayes local se entrena utilizando únicamente las instancias de entrenamiento que satisfacen el antecedente de la regla. La clasificación de la instancia de prueba se realiza utilizando este clasificador local.
Funcionamiento del algoritmo:

- El algoritmo comienza inicializando una regla vacía y un clasificador Naive Bayes entrenado con todos los datos de entrenamiento.
- A continuación, se itera sobre los atributos no incluidos en la regla, evaluando si agregar una condición basada en el valor del atributo en la instancia de prueba puede mejorar la precisión del clasificador.
- Si se encuentra un atributo que mejora la precisión, se actualiza la regla, el clasificador local y el error.
- El proceso continúa hasta que no se puedan encontrar más mejoras.
- Finalmente, se clasifica la instancia de prueba utilizando el clasificador local.

Logistic Models Tree. El algoritmo Logistic Models Tree (LMT) presentado por Landwehr et al. (2005), combina la capacidad de los árboles de decisión para modelar relaciones no lineales en los datos con la robustez de la regresión logística para producir modelos predictivos precisos e interpretables.
LMT construye un árbol de decisión en el que cada hoja (nodo terminal) contiene un modelo de regresión logística en lugar de una simple predicción de clase. Esto se logra mediante un proceso iterativo que combina la inducción del árbol con el ajuste de modelos logísticos:

1. Crecimiento del árbol:
    - Empieza construyendo un modelo de regresión logística en el nodo raíz del árbol utilizando el algoritmo LogitBoost.
    - LogitBoost ajusta el modelo de manera iterativa, agregando funciones de regresión lineal simples hasta que se alcanza un ajuste satisfactorio, determinado por validación cruzada.
    - Se busca el mejor atributo para dividir los datos y se crean nodos hijos correspondientes a los subconjuntos de datos.
2. Construcción de los modelos logísticos:
    - En lugar de entrenar un modelo desde cero en cada nodo hijo, LMT refina el modelo del nodo padre, agregando funciones de regresión lineal.
    - Este proceso de refinamiento iterativo continúa a medida que el árbol crece, lo que resulta en modelos logísticos en las hojas que incorporan la información de los niveles superiores del árbol.
3. Poda del árbol:
    - Para evitar el sobreajuste, LMT poda el árbol utilizando el algoritmo de poda CART.
    - Este algoritmo utiliza validación cruzada para evaluar el rendimiento del árbol con diferentes niveles de poda y selecciona la estructura que minimiza el error de predicción.

Hybrid Fuzzy Decision Trees. El algoritmo Hybrid Fuzzy Decision Tree presentado por Janikow (1998), combinan la lógica difusa, que permite manejar la incertidumbre y la imprecisión en los datos al trabajando con valores de verdad que pueden variar entre 0 y 1, con los árboles de decisión para mejorar la toma de decisiones en ingeniería. Los FDT usan conjuntos difusos para manejar la incertidumbre en los datos y permiten que las muestras se propaguen por múltiples caminos en el árbol con diferentes grados de confianza.
El desarrollo de un FDT implica las siguientes fases:

- Difusión de los datos: Los atributos numéricos se transforman en conjuntos difusos mediante funciones de pertenencia, capturando la incertidumbre inherente a la medición.
- Construcción del Árbol de Decisión Difuso: Se emplean algoritmos de inducción de árboles de decisión, adaptando los criterios de selección de atributos para manejar la naturaleza difusa de los datos.
- Conversión a Reglas Difusas: El árbol de decisión difuso se traduce en un conjunto de reglas difusas del tipo "SI-ENTONCES", expresando el conocimiento de clasificación de forma legible.
- Inferencia Difusa: Se implementa un mecanismo de razonamiento difuso para combinar la información de las reglas activadas y la información del nuevo caso a clasificar.

3. Conclusiones y líneas futuras

En este trabajo, se ha realizado un análisis exhaustivo de los metaclasificadores más representativos. Cada uno de estos enfoques ha demostrado ser valioso en diferentes contextos, destacándose por sus fortalezas específica.

Bagging ha demostrado ser una técnica robusta para mejorar la estabilidad y precisión de los clasificadores, particularmente en escenarios donde los modelos base son sensibles a la varianza. Su capacidad para generar conjuntos diversos mediante el muestreo con reemplazo y combinar sus predicciones mediante votación ha sido clave para su efectividad. No obstante, su dependencia de clasificadores base potentes y el balance entre el tamaño del conjunto y el coste computacional son áreas que aún pueden optimizarse. Además, la falta de atención a la evaluación dinámica de su rendimiento en tiempo real sugiere posibles líneas futuras de investigación para maximizar su aplicabilidad en entornos más exigentes.

Por otra parte, Boosting es capaz de convertir clasificadores débiles en modelos fuertes mediante la combinación iterativa y adaptativa de predicciones. Algoritmos como AdaBoost, Gradient Boosting y sus variantes han demostrado gran eficacia en tareas de clasificación, aunque tiene limitaciones, como evitar el sobreajuste y su falta de escalabilidad en coste computacional.

En la investigación de Random Forest se ha enfocado en mejorar clasificadores base optimizando su rendimiento, diversidad y combinación. Sin embargo, aspectos como el mecanismo de votación y la evaluación mediante datos fuera de bolsa (OOB) han recibido menos atención, lo que puede generar resultados inconsistentes. Además, aunque se han explorado estrategias para aumentar la diversidad entre árboles, la falta de una cuantificación clara de esta limita la solidez de los resultados. La integración equilibrada de clasificación y diversidad sigue siendo un desafío pendiente.

Los clasificadores en cascada son herramientas eficaces para tareas de clasificación complejas, combinando simplicidad y precisión mediante un enfoque progresivo. Su arquitectura jerárquica permite resolver rápidamente instancias sencillas con clasificadores básicos y delegar los casos más difíciles a etapas posteriores más complejas, optimizando el costo computacional. Avances, como boosting cascade, soft cascade y configuraciones automáticas, han mejorado su robustez, flexibilidad y capacidad de adaptación a diversos problemas. En el futuro, las investigaciones podrían enfocarse en integrar cascadas con redes neuronales profundas, optimizar su diseño mediante aprendizaje automatizado, adaptarlas a big data y flujos continuos, y mejorar su interpretabilidad y capacidad multitarea, abordando además sesgos presentes en los datos y modelos.

Los clasificadores híbridos, representan una estrategia eficaz para mejorar la precisión y robustez de los modelos al combinar algoritmos diversos que aprovechan sus fortalezas y compensan sus debilidades no solo en tareas de clasificación, sino también por su adaptabilidad a diferentes dominios y datasets.

Sin embargo, cada uno presenta desafíos específicos, como la necesidad de ajustar hiperparámetros o la elevada complejidad computacional en algunos casos y la selección adecuada de los clasificadores base y de la integración efectiva entre ellos. Además, una limitación importante en estos métodos es su falta de interpretabilidad. Por tanto, como líneas futuras se podría desarrollar enfoques que hagan más transparentes las decisiones de los metaclasificadores, utilizando técnicas como explicaciones basadas en importancia, análisis de contribución local o métodos post-hoc como SHAP y LIME, como primera propuesta. Por otro lado, investigar técnicas para reducir los costos computacionales asociados a métodos como Boosting y Rotation Forest, haciéndolos más accesibles en aplicaciones en tiempo real. Además, su integración con tecnologías emergentes podría ampliar sus aplicaciones en problemas dinámicos y complejos.

Referencias

Alpaydın, E. and Kaynak, C. (1997). Cascading classifiers. *Kybernetika*, 34:369–374.

Bartlett, P., Freund, Y., Lee, W. S., and Schapire, R. E. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *The annals of statistics*, 26(5):1651–1686.

Bentéjac, C., Csörgő, A., and Martínez-Muñoz, G. (2021). A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, 54:1937–1967.

Bielza, C. and Larrañaga, P. (2020). *Data-Driven Computational Neuroscience: Machine Learning and Statistical Models*. Cambridge University Press.

Bourdev, L. and Brandt, J. (2005). Robust object detection via soft cascade. volume 2, pages 236–243 vol. 2.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

Breiman, L. (1997). Arcing the edge. Technical report, Citeseer.

Breiman, L. (1998). Arcing classifier (with discussion and a rejoinder by the author). *The annals of statistics*, 26(3):801–849.

Breiman, L. (1999a). Pasting small votes for classification in large databases and on-line. In *Machine Learning*, pages 85–103. Springer.

Breiman, L. (1999b). Using adaptive bagging to debias regressions. Technical Report 547, Statistics Department, University of California at Berkeley, Berkeley, CA 94720.

Breiman, L. (2001). Random forests. *Machine learning*, 45:5–32.

Bühlmann, P. and Yu, B. (2002). Analyzing bagging. *The Annals of Statistics*, 30(4):927–961.

Bühlmann, P. and Yu, B. (2003). Bagging, subagging and bragging for improving some prediction algorithms. *Statistical Science*, 18(4):424–438.

Chen, C., Liaw, A., and Breiman, L. (2004). Using random forest to learn imbalanced data. dept. statistics. *Univ. California, Berkeley, CA, Tech. Rep*, 666.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794.

Collins, M., Schapire, R. E., and Singer, Y. (2002). Logistic regression, adaboost and bregman distances. *Machine Learning*, 48:253–285.

Delgado-Panadero, , Benítez-Andrades, J. A., and García-Ordás, M. T. (2023). A generalized decision tree ensemble based on the neuralnetworks architecture: Distributed gradient boosting forest (dgbf). *Applied Intelligence*, 53(19):22991–23003.

Demiriz, A., Bennett, K. P., and Shawe-Taylor, J. (2002). Linear programming boosting via column generation. *Machine Learning*, 46:225–254.

Denil, M., Matheson, D., and De Freitas, N. (2014). Narrowing the gap: Random forests in theory and in practice. In *International conference on machine learning*, pages 665–673. PMLR.

Emami, S. and Martínez-Muñoz, G. (2024). Condensed-gradient boosting. *International Journal of Machine Learning and Cybernetics*, pages 1–15.

Farhadi, Z., Bevrani, H., and Feizi-Derakhshi, M.-R. (2024). Improving random forest algorithm by selecting appropriate penalized method. *Communications in Statistics-Simulation and Computation*, 53(9):4380–4395.

Fdez-Díaz, L., Quevedo, J. R., and Montañés, E. (2023). Regularized boosting with an increasing coefficient magnitude stop criterion as meta-learner in hyperparameter optimization stacking ensemble. *Neurocomputing*, 551:126516.

Florek, P. and Zagdański, A. (2023). Benchmarking state-of-the-art gradient boosting algorithms for classification.

Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Information and computation*, 121(2):256–285.

Freund, Y. (1999). An adaptive version of the boost by majority algorithm. In *Proceedings of the twelfth annual conference on Computational learning theory*, pages 102–113.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232.

Friedman, J. H. (2002). Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378.

Gabidolla, M. and Carreira-Perpiñán, M. Á. (2022). Pushing the envelope of gradient boosting forests via globally-optimized oblique trees. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 285–294.

Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine learning*, 63:3–42.

Han, S., Kim, H., and Lee, Y.-S. (2020). Double random forest. *Machine Learning*, 109:1569–1586.

Hastie, T., Rosset, S., Zhu, J., and Zou, H. (2009). Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360.

Ho, T. K. (1995). Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE.

Ho, T. K. (1998). The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844.

Janikow, C. Z. (1998). Fuzzy decision trees: issues and methods. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 28(1):1–14.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30.

Kearns, M. and Valiant, L. (1994). Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95.

Kohavi, R. et al. (1996). Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Kdd*, volume 96, pages 202–207.

Kuncheva, L. I. (2014). *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons.

Landwehr, N., Hall, M., and Frank, E. (2005). Logistic model trees. *Machine learning*, 59:161–205.

Li, B., Yu, Q., and Peng, L. (2022). Ensemble of fast learning stochastic gradient boosting. *Communications in Statistics-Simulation and Computation*, 51(1):40–52.

Li, Y., Yan, C., Liu, W., and Li, M. (2018). A principle component analysis-based random forest with the potential nearest neighbor method for automobile insurance fraud identification. *Applied Soft Computing*, 70:1000–1009.

Luo, J., Wei, Z., Man, J., and Xu, S. (2023). Trboost: a generic gradient boosting machine based on trust-region method. *Applied Intelligence*, 53(22):27876–27891.

Mason, L., Baxter, J., Bartlett, P., and Frean, M. (1999). Boosting algorithms as gradient descent. In Solla, S., Leen, T., and Müller, K., editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press.

Menze, B. H., Kelm, B. M., Masuch, R., Himmelreich, U., Bachert, P., Petrich, W., and Hamprecht, F. A. (2009). A comparison of random forest and its gini importance with standard chemometric methods for the feature selection and classification of spectral data. *BMC bioinformatics*, 10:1–16.

Mukherjee, I. and Schapire, R. E. (2010). A theory of multiclass boosting. *Advances in Neural Information Processing Systems*, 23.

Oza, N. C. and Russell, S. J. (2001). Online bagging and boosting. In Richardson, T. S. and Jaakkola, T. S., editors, *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics*, volume R3 of *Proceedings of Machine Learning Research*, pages 229–236. PMLR. Reissued by PMLR on 31 March 2021.

Ponomareva, N., Radpour, S., Hendry, G., Haykal, S., Colthurst, T., Mitrichev, P., and Grushetsky, A. (2017). Tf boosted trees: A scalable tensorflow based framework for gradient boosting. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18–22, 2017, Proceedings, Part III 10*, pages 423–427. Springer.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. (2018). Catboost: unbiased boosting with categorical features. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Pudil, P., Novovicova, J., Blaha, S., and Kittler, J. (1992). Multistage pattern recognition with reject option. In *Proceedings., 11th IAPR International Conference on Pattern Recognition. Vol.II. Conference B: Pattern Recognition Methodology and Systems*, pages 92–95.

Rätsch, G., Warmuth, M. K., and Glocer, K. (2007). Boosting algorithms for maximizing the soft margin. *Advances in neural information processing systems*, 20.

Rodriguez, J. J., Kuncheva, L. I., and Alonso, C. J. (2006). Rotation forest: A new classifier ensemble method. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1619–1630.

Rokach, L. (2010). Ensemble-based classifiers. *Artificial intelligence review*, 33:1–39.

Saberian, M. and Vasconcelos, N. (2010). Boosting classifier cascades. pages 2047–2055.

Saffari, A., Leistner, C., Santner, J., Godec, M., and Bischof, H. (2009). On-line random forests. In *2009 ieee 12th international conference on computer vision workshops, iccv workshops*, pages 1393–1400. IEEE.

Schapire, R. E. (2003). The boosting approach to machine learning: An overview. *Nonlinear estimation and classification*, pages 149–171.

Tripoliti, E. E., Fotiadis, D. I., and Manis, G. (2013). Modifications of the construction and voting mechanisms of the random forests algorithm. *Data & Knowledge Engineering*, 87:41–65.

Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I.

Wallace, B. C., Small, K., Brodley, C. E., and Trikalinos, T. A. (2011). Class imbalance, redux. In *2011 IEEE 11th International Conference on Data Mining*, pages 754–763. IEEE.

Wang, Y., Xia, S.-T., Tang, Q., Wu, J., and Zhu, X. (2017). A novel consistent random forest framework: Bernoulli random forests. *IEEE transactions on neural networks and learning systems*, 29(8):3510–3523.

Wold, S., Esbensen, K., and Geladi, P. (1987). Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52.

Wolpert, D. H., Macready, W. G., et al. (1995). No free lunch theorems for search. Technical report, Citeseer.

Woźniak, M., Grana, M., and Corchado, E. (2014). A survey of multiple classifier systems as hybrid systems. *Information Fusion*, 16:3–17.

Xiao, R. and Zhu, L. (2003). Boosting chain learning for object detection. volume 1, pages 709–715 vol.1.

Ye, J., Janardan, R., and Li, Q. (2004). Two-dimensional linear discriminant analysis. *Advances in neural information processing systems*, 17.

Zhang, C.-X. and Zhang, J.-S. (2008). Rotboost: A technique for combining rotation forest and adaboost. *Pattern recognition letters*, 29(10):1524–1536.

Zhang, C.-X. and Zhang, J.-S. (2010). A variant of rotation forest for constructing ensemble classifiers. *Pattern Analysis and Applications*, 13:59–77.

Zhang, L. and Suganthan, P. N. (2014). Random forests with ensemble of feature spaces. *Pattern Recognition*, 47(10):3429–3437.

Zhang, Z. and Jung, C. (2020). Gbdt-mo: gradient-boosted decision trees for multiple outputs. *IEEE transactions on neural networks and learning systems*, 32(7):3156–3167.

Zheng, Z. and Webb, G. I. (2000). Lazy learning of bayesian rules. *Machine learning*, 41:53–84.